FICHEROS

Se llama a los datos que se guardan en ficheros datos persistentes, porque persisten más allá de la ejecución de la aplicación que los trata. Los ordenadores almacenan los ficheros en unidades de almacenamiento secundario como discos duros, discos ópticos, etc.

A las operaciones, que constituyen un flujo de información del programa con el exterior, se les conoce como Entrada/Salida (E/S).

Un archivo o fichero es una colección de datos homogéneos almacenados en un soporte físico del pc que puede ser permanente o volátil.

- Datos homogéneos: almacena colecciones de datos del mismo tipo (igual que arrays)
- Cada elemento almacenado en un fichero se denomina registro, que se compone de campos.

Operaciones sobre ficheros

- Operación de creación (sólo si no existe previamente)
- Operación de apertura. Varios modos:
    - Sólo lectura.
    - Sólo escritura.
    - Lectura y escritura.
- Operaciones de lectura/escritura.
- Operaciones de inserción/borrado
- Operaciones de desplazamiento dentro de un fichero.
- Operaciones de cierre.

Manejo habitual de un fichero

1. Crearlo
2. Abrirlo
3. Operar sobre él (lectura/escritura, inserción, borrado, etc).
4. Cerrarlo.

Tipos de ficheros

Clasificación según la organización de los registros en memoria:

- Organización secuencial: registros almacenados consecutivamente en memoria según el orden lógico en que se han ido insertando.
- Organización directa o aleatoria: el orden físico de almacenamiento en memoria puede no coincidir con el orden en que han sido insertados.
- Organización indexada. Dos ficheros (de datos y de índice: contiene la posición de cada uno de los registros en el fichero de datos).

Clasificación de los ficheros según el acceso a la información almacenada:

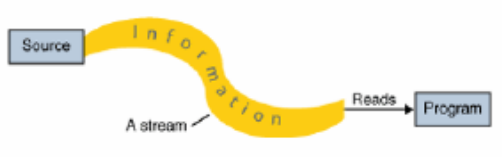
- Acceso secuencial: para acceder a un registro es necesario pasar por todos los anteriores. Ej: cinta de casete.
- Acceso directo o aleatorio: se puede acceder a un registro sin pasar por todos los anteriores. Ej: disco duro.

Clasificación de los ficheros según el tipo de información almacenada:

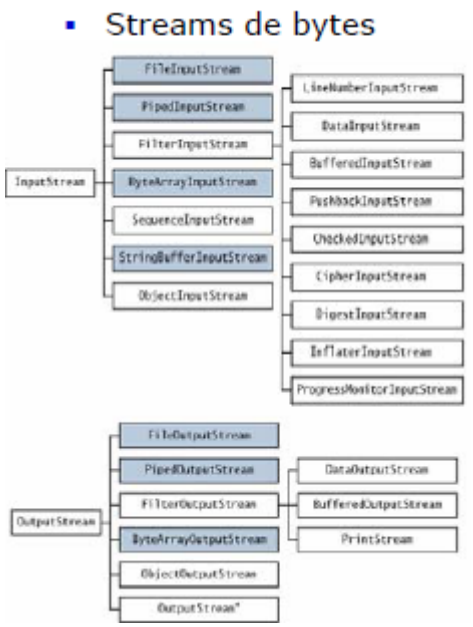
- Ficheros binarios: almacenan secuencias de dígitos binarios (ej: ficheros que almacenan enteros, floats, …)
- Ficheros de texto: almacenan caracteres alfanuméricos en formato estándar (ASCII, Unicode, UTF8, UTF16, etc). Pueden ser leídos y/o modificados por aplicaciones denominadas editores de texto (Ej: Notepad…).

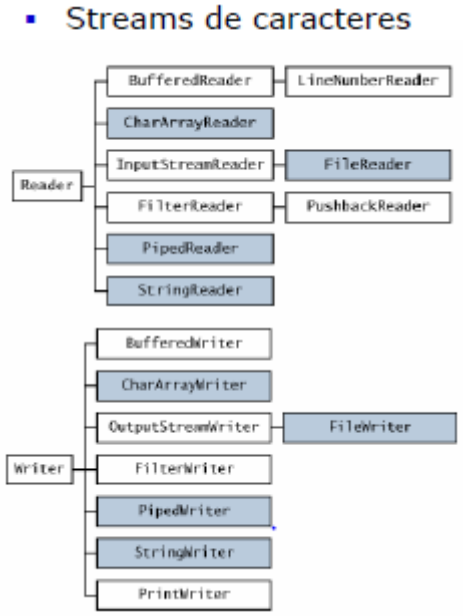
Conceptos básicos Entrada/salida

- Stream: canales, flujos de datos o “tuberías” .
- Entrada (InputStream) o Salida (OutputStream)



- Agrupados en el paquete java.io. Se agrupan las operaciones de E/S en Java que incorporan interfaces, clases y excepciones para acceder a todo tipo de ficheros.
- Dos jerarquías de clases independientes, una para lectura/escritura binaria (bytes) y otra para lectura/escritura de caracteres de texto (char).





Métodos básicos de **Reader**:

int read()

int read(char cbuf[])

nt read(char cbuf[], int offset, int length)

Métodos básicos de **InputStream**:

int read()

int read(byte cbuf[])

int read(byte cbuf[], int offset, int length)

Métodos básicos de **Writer**:

int write(int c)

int write(char cbuf[])

int write(char cbuf[], int offset, int length)

Métodos básicos de **OutputStream**:

int write(int c)

int write(byte cbuf[])

int write(byte cbuf[], int offset, int length)

CLASES ASOCIADAS A OPERACIONES DE GESTIÓN DE FICHEROS Y DIRECTORIOS

CLASE FILE

Nos permite hacer unas cuantas operaciones con ficheros. La clase **File** proporciona una representación abstracta de ficheros y directorios.

Las instancias de la clase **File** representan nombres de archivo. Un objeto de la clase **File** permite examinar el nombre del archivo, descomponerlo en su rama de directorios o crear el archivo si no existe, pasando el objeto de tipo **File** a un constructor adecuado como FileWriter (File f), que recibe como parámetro el objeto File.

Para archivos que existen, a través del objeto File, un programa puede examinar los atributos del archivo, cambiar su nombre, borrarlo o cambiar sus permisos. Dado un objeto File, podemos hacer las siguientes operaciones con él:

- Renombrar el archivo, con el método **renameTo**(). El objeto File dejará de referirse al archivo renombrado, ya que el String con el nombre del archivo en el objeto File no cambia.
- Borrar el archivo, con el método **delete**(). También, con **deleteOnExit**() se borra cuando finaliza la ejecución de la máquina virtual Java.
- Crear un nuevo fichero con un nombre único. El método estático **createTempFile**() crea un fichero temporal y devuelve un objeto File que apunta a él. Es útil para crear archivos temporales, que luego se borran, asegurándonos tener un nombre de archivo no repetido.

- Establecer la fecha y la hora de modificación del archivo con **setLastModified**(). Por ejemplo, se podría hacer: new File("prueba.txt").**setLastModified**(new Date().**getTime**()); para establecerle la fecha actual al fichero que se le pasa como parámetro, en este caso prueba.txt.
- Crear un directorio, mediante el método **mkdir**(). También existe **mkdirs**(), que crea los directorios superiores si no existen.
- Listar el contenido de un directorio. Los métodos **list**() y **listFiles**() listan el contenido de un directorio. **list**() devuelve un vector de String con los nombres de los archivos, **listFiles**() devuelve un vector de objetos File.
- Listar los nombres de archivo de la raíz del sistema de archivos, mediante el método estático **listRoots**().

INTERFACE FILENAMEFILTER

Hemos visto cómo obtener la lista de ficheros de una carpeta o directorio. A veces, nos interesa ver no la lista completa, sino los archivos que encajan con un determinado criterio.

Por ejemplo, nos puede interesar un filtro para ver los ficheros modificados después de una fecha, o los que tienen un tamaño mayor del que indiquemos, etc.

El interface FilenameFilter se puede usar para crear filtros que establezcan criterios de filtrado relativos al nombre de los ficheros. Una clase que lo implemente debe definir e implementar el método:

```
boolean accept(File dir, String nombre)
```

Este método devolverá verdadero en el caso de que el fichero cuyo nombre se indica en el parámetro nombre aparezca en la lista de los ficheros del directorio indicado por el parámetro dir.

En el siguiente ejemplo vemos cómo se listan los ficheros de la carpeta C:\datos que tengan la extensión .txt. Usamos try y catch para capturar las posibles excepciones, como que no exista dicha carpeta.

```
import java.io.File;
import java.io.FilenameFilter;

public class Filtrar implements FilenameFilter {

    String extension;

    // Constructor
    Filtrar(String extension){

        this.extension = extension;
    }

    public boolean accept(File dir, String name){

        return name.endsWith(extension);
    }

    public static void main(String[] args) {

        try {

            // Obtendremos el listado de los archivos de ese directorio
            File fichero = new File("c:\\datos\\.");
            String[] listadeArchivos = fichero.list();
            // Filtraremos por los de extension .txt
            listadeArchivos = fichero.list(new Filtrar(".txt"));
            // Comprobamos el número de archivos en el listado
            int numarchivos = listadeArchivos.length ;
```

```
            // Si no hay ninguno lo avisamos por consola
            if (numarchivos< 1) {

                System.out.println("No hay archivos que listar");

            // Y si hay, escribimos su nombre por consola.
            else {

                for(intconta = 0; conta<listadeArchivos.length; conta++)

                    System.out.println(listadeArchivos[conta]);
            }

        } catch (Exception ex) {

            System.out.println("Error al buscar en la ruta indicada");
        }
    }
}
```

CREACIÓN Y ELIMINACIÓN DE FICHEROS Y DIRECTORIOS

Cuando creamos un fichero, podemos hacerlo del siguiente modo:

```
try {

    // Creamos el objeto que encapsula el fichero
    File fichero = new File("c:\\prufba\\miFichero.txt");

    // A partir del objeto File creamos el fichero físicamente
    if (fichero.createNewFile()) {

    System.out.println("El fichero se ha creado correctamente");

} else {

    System.out.println("No ha podido ser creado el fichero");

} catch (Exception ioe) {

    ioe.getMessage();
}
```

Para borrar un fichero podemos usar la clase File, comprobando previamente si existe el fichero, del siguiente modo:

```
File fichero = new File("C:\\prueba", "agenda.txt"); if

(fichero.exists())

    fichero.delete();
```

Para crear directorios, podríamos hacer:

```
try {

    // Declaración de variables String
    directorio = "C:\\prueba";

    String varios = "carpeta1/carpeta2/carpeta3";

    // Crear un directorio

    boolean exito = (new File(directorio)).mkdir();

    if (exito)

        System.out.println("Directorio: " + directorio + " creado");

    // Crear varios directorios

    exito = (new File(varios)).mkdirs();

    if (exito)
```

```
            System.out.println("Directorios: " + varios + " creados");

        }catch (Exception e){

            System.err.println("Error: " + e.getMessage());
        }
```

Para borrar un directorio con Java, tendremos que borrar cada uno de los ficheros y directorios que éste contenga. Al poder almacenar otros directorios, se podría recorrer recursivamente el directorio para ir borrando todos los ficheros.

Se puede listar el contenido del directorio e ir borrando con:

```
File[] ficheros = directorio.listFiles();
```

Si el elemento es un directorio, lo sabemos mediante el método isDirectory().

CREACIÓN DE FICHEROS EN ECLIPSE

Si no se indica ruta cuando le das el nombre el constructor, el fichero se creará en el proyecto donde está esa clase.

Para poder ver el fichero, click derecho en el proyecto y “Refresh”.